



(11) **EP 1 022 908 A1**

(12) **EUROPEAN PATENT APPLICATION**

(43) Date of publication:  
**26.07.2000 Bulletin 2000/30**

(51) Int Cl.7: **H04N 7/24**

(21) Application number: **99300438.1**

(22) Date of filing: **21.01.1999**

(84) Designated Contracting States:  
**AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU**  
**MC NL PT SE**  
 Designated Extension States:  
**AL LT LV MK RO SI**

(71) Applicant: **Sony Service Center (Europe) N.V.**  
**1840 Londerzeel (BE)**

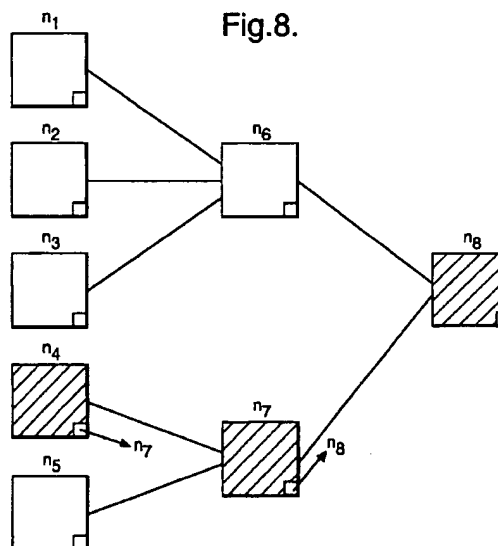
(72) Inventors:  
 • **Rademakers, Phillip,**  
**Sony Service Ctr. (Europe) NV**  
**1130 Bruxelles (BE)**  
 • **Cuypers, Ludo,**  
**Sony Service Center (Europe) N.V.**  
**1130 Bruxelles (BE)**  
 • **Wasowski, Maciek,**  
**Sony Service Ctr. (Europe) N.V.**  
**1130 Bruxelles (BE)**

• **Heughebaert, André,**  
**Sony Serv. Ctr. (Europe) N.V.**  
**1130 Bruxelles (BE)**  
 • **Baranluk, Monika,**  
**Sony Service Ctr. (Europe) N.V.**  
**1130 Bruxelles (BE)**  
 • **Mitchell, Joe, Sony Service Center (Europe) N.V.**  
**1130 Bruxelles (BE)**  
 • **Aerts, Ives,**  
**c/o Sony Service Center (Europe) N.V.**  
**1130 Bruxelles (BE)**

(74) Representative: **Ayers, Martyn Lewis Stanley**  
**J.A. KEMP & CO.**  
**14 South Square**  
**Gray's Inn**  
**London WC1R 5LX (GB)**

(54) **Information server and method of constructing a transport stream**

(57) An information server for outputting a carousel files from a transport stream and a method of constructing a transport stream from a carousel of files wherein data from the files is transformed through a protocol stack to transport stream packets and, for each file used in a transformation, a dependency link is stored indicating the transformed data resulting from the transformation and, for any transformed data used in a transformation to form further transformed data, a dependency link is stored indicating the further transformed data, such that, when a file in the carousel is changed, the dependency links indicate any transformed data requiring recalculation as a result of the change.



**Description**

[0001] The present invention relates to an information server and a method of constructing a transport stream, in particular to a dependency network and a method of transforming data from carousel files into transport stream packets in which dependency links are established for the transformation.

[0002] It has been proposed to provide an information server for providing information in selected channels of a transport stream. For instance, various data files may be provided in particular channels of a transport stream so as to broadcast video data and such like. It is desired that a number of different data files are transmitted in sequence by way of a carousel. It is also desired that individual files of the carousel should be changed at times. Unfortunately, however, because of the transformation of the files through a protocol stack to form transport stream packets, there is no direct correspondence between the original files and the transport stream packets. Hence, replacement of individual files results in more significant changes in the transport stream.

[0003] According to the present invention, there is provided a method of constructing a transport stream from a carousel of files, the method comprising:

transforming data from the files through a protocol stack to transport stream packets;  
for each file used in a transformation, storing a dependency link indicating the transformed data resulting from the transformation and, for any transformed data used in a transformation to form further transformed data, storing a dependency link indicating the further transformed data, such that, when a file in the carousel is changed, the dependency links indicate any transformed data requiring recalculation as a result of the change.

[0004] According to the present invention, there is also provided an information server for outputting a carousel of files in a transport stream, the information server comprising:

at least one carousel builder for transforming data from the files through a protocol stack to form transport stream packets;  
a series of registers; and  
a dependency mechanism which, for each file used in a transformation, stores a dependency link indicating the transformed data resulting from the transformation and, for any transformed data used in a transformation to form further transformed data, stores a dependency link indicating the further transformed data such that, when a file in the carousel is changed, the dependency links stored in the registers indicate any transformed data requiring recalculation as a result of the change.

[0005] In this way, when a file is changed or replaced, a simple structure is provided to indicate only those portions of the overall transformation for the transport stream which require recalculation. In other words, when files are changed, it is not necessary to recalculate transformations for the entire transport stream.

[0006] For a large carousel, recalculation of the entire corresponding transport stream would take an unacceptable length of time. However, by identifying only those portions of the transport stream which require recalculation, it is possible to provide an information server which may quickly update individual files in a carousel.

[0007] Preferably, the registers form part of a dependency network comprising dependency nodes, the dependency nodes including primary nodes that are based on files of the carousel and computed nodes that are based on transformed data.

[0008] The dependency nodes preferably store pointers to particular parts of the bit stream representation of the carousel of files and the dependency links indicate how one set of pointers is used to calculate another set of pointers, the pointers acting as content references.

[0009] In this way, the dependency network provides a computationally straightforward mechanism for the information server to identify the required portions of the bit stream for insertion into the transport stream. By following the dependency structure, pages in the original files can easily be accounted for.

[0010] Preferably, the carousel comprises a DSMCC object carousel, such that the content reference pointers of the dependency nodes point to particular sub-parts of the bit stream representation of the DSMCC object carousel and the primary nodes contain the content references of the DSMCC file and directory objects.

[0011] Once again, this provides a computationally straightforward system for identifying appropriate data for insertion into the transport stream packets.

[0012] At least one of the files of the carousel may comprise different versions of that file, such that a different version may be transmitted on consecutive rounds of the carousel broadcast. In this case, the at least one carousel builder preferably transforms the data of each of said versions and the dependency structure stores the dependency links for each version, such that consecutive versions of the transport stream packets are selected upon each round of the carousel broadcast.

[0013] In this way, upon each round of the carousel broadcast, it is not necessary for the information server to recalculate the required payloads and transmission stream packets. The dependency structure allows the information server merely to change a pointer with regard to the different versions, such that the output transport stream is easily changed upon each round of the carousel.

[0014] The present invention will be more clearly understood from the following description, given by way of example only, with reference to the accompanying drawings, in which:

Figure 1 illustrates a broadcasting network in which the present invention may be embodied;

Figure 2 illustrates the play-out section of an Information Server in which the present invention may be embodied;

Figure 3 illustrates an arrangement of Information Servers in which the present invention may be embodied;

Figures 4 A and B illustrate an Information Server in which the present invention may be embodied;

Figure 5 illustrates an object/data carousel;

Figure 6 illustrates an array of notional time slots for prioritising data operations;

Figure 7 illustrates the construction of DVB sections from files; and

Figure 8 illustrates a dependency structure for data transformations.

[0015] The following description relates to an Information Server to which the present invention may be applied. In particular, an architecture is proposed for a DVB compliant data server, called Information Server, that is capable of generating DVB compliant transport streams containing DSMCC Data and Object Carousels. In addition, the Information Server is to support the dynamic updating of content inside a DSMCC Data and Object Carousel while it is being streamed out.

[0016] Figure 1 illustrates part of a DVB system in which the Information Server is incorporated. However, it should be appreciated that the Information Server can also be incorporated in other DVB networks, for instance cable or terrestrial, and can also be incorporated in other non-DVB networks.

[0017] A play-out section 2 may incorporate a number of video/audio sources 4, as well as live sources 6. Data from these sources, for instance in the form of MPEG2 transport stream packets, is fed to a multiplexer 8 and output from the play-out section 2. The multiplexed stream of data is fed to a modulator 10 for transmission, in this example, from a satellite 12. The signal is then received by the set-up box 14 of a television 16. Signals from a number of other modulators 10', 10" etc. may be fed together and transmitted to the satellite 12.

[0018] In this system, it is also possible to provide an Information Server 18. Just like the video/audio sources 4, 6, the Information Server 18 provides data, for instance in the form of MPEG2 transport stream packets to the multiplexer 8 for insertion into the data stream for transmission. The Information Server may be used to broadcast any form of data, but, in particular, MHEG 5 applications, HTML pages, Java applications, IRD software upgrades and other data services in the form of DSMCC objects on data carousels. As illustrated, the Information Server 18 is provided with a host 20 which is used to control the behaviour of the Information Server 18, together with the content and nature of the data output from the Information Server 18.

[0019] The Information Server is controlled by the host computer using some control network such as TCP/IP over ethernet. The generated transport streams are played out over some streaming network such as ethernet, DVB ASI, DVB LVDS or ATM to the receivers (PCs, STBs etc.).

[0020] Figure 2 illustrates the overall architecture of the Information Server.

[0021] An API server 22 receives control and data from an external host. Data is transferred to a storage unit 24 for subsequent retrieval. At least one Stream Generator 26 is provided for outputting an MPEG2 data stream. Thus, in operation, the API server receives data, for instance a series of images, and transfers the data of these images to the storage 24. Each of the Stream Generators 26 generates a stream of data under the control of the API server 22 based on a description of the carousel. In particular, the API server 22 receives control from the external host and, in turn, controls the Stream Generators 26 appropriately. As will be described later, the Stream Generators 26 may comprise carousel builders so as to transmit data from the storage 24 in the correct sequence.

[0022] The purpose of the API server is to provide a software interface that is remotely accessible to a host computer (e.g. a PC) over TCP/IP. The software interface is realised by using an RPC protocol and provides the user of the Information Server with a set of software abstractions that allow the user to define object and data carousels in a way that is independent of the transport bit stream but that rather focus on the conceptual structure of the carousels starting

from the content that must be broadcast by the carousels.

**[0023]** In addition, the API server manages the content data which is present on the content disk of the Information Server.

**[0024]** From an architectural point of view, the API server is realised as a single threaded process containing two major modules: "RPC layer" and "API Implementation". The "RPC layer" manages the control network connection with one or multiple host computers. Each connection is called a session. It receives the RPC requests from the host computer session, serialises the requests, checks whether they are valid, demarshalls the data (arguments) for the requests, invokes an appropriate entry point in the "API Implementation", receives the result (or error) from the API Implementation, marshalls the received results into an RPC reply and finally sends the reply back to the host computer. The API Implementation provides a set of functions that implement the functionality of the Information Server API. It interacts with the content disk, reads/writes data to this disk and checks the validity of the arguments in each API call.

**[0025]** Transmission of the streams from the Stream Generators 26 to the multiplexer of the play out section 2 may take a number of different forms. It is proposed that the streaming be provided as a separate unit to the Information Server. Thus, streaming may be chosen from, for instance, DVB-ASI via a PRISM board (as produced by Sony Digital Network Solutions Japan), DVB-LVDS, ATM or Ethernet.

**[0026]** It is proposed that the Information Server be provided as a standard 4U 19 inch rack-mount. The control network between the host computer and the Information Server might be, for instance, Ethernet or ATM.

**[0027]** Figure 3 illustrates how one host computer may be used to control any number of Information Servers, each producing a number of streams.

**[0028]** Figure 4A illustrates further details of the Information Server with reference to only one Stream Generator. The API server 22 includes the RPC (Remote Procedure Call) protocol allowing the APIs to be accessed by the host computer. It is also possible that the control APIs may run with IIO as this becomes more popular. Thus, the API server 22 receives data and control from the remote host computer. The API server 22 also runs under an operating system abstraction layer or OSAL. This is provided to allow the Information Server to be run on a number of different operating systems. In particular, in order to adapt the Information Server to run on a different operating system, it is only necessary to change the OSAL.

**[0029]** The Stream Generator 26 is capable of outputting raw transport stream data. In particular, under the command of the API server 22, the Stream Generator 26 merely retrieves data from the storage 24 and outputs it without any knowledge of its content. However, it is also intended that the Stream Generator 26 be capable of constructing carousels, for instance DSMCC object carousels and DSMCC data carousels. Hence, the Stream Generator 26 includes at least one carousel builder 28.

**[0030]** Figure 4B illustrates a Stream Generator. The Stream Generator is responsible for dynamically building the transport bit stream starting from the conceptual description of the Data/Object Carousel. The architecture of the Stream Generator consists of a number of concurrently running threads. There are the following threads:

1. SGMain: this thread is responsible for communicating with the API server. It receives commands from the API server to start or stop an object carousel, to drop a carousel from a running stream, or to update some content inside a running stream. In addition, the SGMain thread is also responsible for managing the other threads. This implies the starting/stopping of these threads and to control their synchronization.

2. Carousel builder (CB): this thread is responsible for constructing the (binary) DSMCC sections that make up the object carousel. The CB thread receives the carousel description from the SGMain thread. Once the DSMCC sections are constructed they are delivered to the Mux thread as payloads (see below). The CB thread also communicates with the SI thread to insert the proper service information in the tables managed by the SI thread.

3. SI: this thread is responsible for generating and updating service information (SI) tables. It dynamically constructs the MPEG2 sections that make up these SI tables and delivers the sections as payloads to the Mux thread. Currently, 2 tables are managed by the SI thread; PAT and PMT. The CB thread receives requests from the CB thread to reserve programme numbers in the PMT, to reserve PIDs for the carousels and to add or delete descriptions for the PMT that are needed to identify the carousels.

4. Mux: this thread receives payloads from the CB and SI thread and multiplexes the payload into MPEG2 transport stream packets. In addition, the Mux thread communicates with the physical device driver to pump out the transport stream packets over the physical interface ethernet, ASI, ATM.

**[0031]** Each component in the Stream Generator (SGMain, SI, Carousel Builder, Mux) is a separate thread running concurrently with the other threads. The SI, Carousel Builder work in "push" mode. They compute parts of the transport bit stream and push them forward to the Mux. The Mux thread works in "pull" mode. As soon as data is available from either the SI or Carousel Builders, it pulls off this data and starts multiplexing the payloads into the transport stream packets.

**[0032]** Each carousel builder 28 is capable of constructing a carousel of information, for instance as illustrated in

Figure 5. The carousel builder constructs transport stream data appropriately for transmitting in rotation a series of DSMCC files. For instance, each page as illustrated in Figure 5 might comprise MHEG or HTML files or Java applications. As a result, the stream produced by the Stream Generator includes each of those pages in turn.

**[0033]** It is also possible for the carousel builder to construct sub-carousels, such that on each rotation of the main carousel of pages, a particular page itself rotates through a series of sub-pages.

**[0034]** Since each output stream may contain a number of different carousels, a number of different carousel builders 28 may also be provided. The multiplexer/packetiser 30 arranges the data from the various carousel builders 28 into the output stream.

**[0035]** As illustrated in Figures 4 A and B, the Stream Generator 26 also uses the Operating System Abstraction Layer. Furthermore, it uses a Driver Abstraction Layer. In this way, as with the operating system, the output driver may be changed without changing the entire construction of the Information Server. Instead, it is merely necessary to change the driver implementation.

**[0036]** The driver 32 delivers transport stream data to a DVB encoder board.

**[0037]** Each of the carousel builders 28 may operate with its own data, the multiplexer handling scheduling between the various carousels of the transport stream.

**[0038]** The carousel builders effectively have their own memories. Indeed, having constructed the pages of the carousel from data in the storage 24, the transport stream data of those pages may be stored by the carousel builders. In this way, when a page of a carousel is changed under the instruction of the API server from the host computer, it may not be necessary to recalculate the data of the entire carousel, but only of the changed page. Even where the data of one page is dependent on other pages, it is also proposed to incorporate a dependency structure, such that, once again, it is only necessary to recalculate data for pages which require change and not all of the pages.

**[0039]** Each carousel builder 28 reads the content and content descriptions, assembles the DSMCC object carousels and prepares appropriate DSI, DII or DDB sections.

**[0040]** The SI tables manager 34 is for managing PAT/PMT sections, programme numbers and PIDs. The multiplexer/packetiser 30 is for splitting the sections into packets and assembling the packets into slots for the driver 32.

**[0041]** As mentioned above, the Information Server preferably includes an operating system abstraction layer OSAL. This may be an object orientated C++ based class library OSAL is proposed that provides an API that abstracts some of the essential operating system services offered by various operating systems such as Windows 95, Windows NT, UNIX, LINUX etc. It is proposed that the OSAL provides abstractions for threads, semaphores, files, directories, processors, file system independent path names and inter-process message queues.

**[0042]** The OSAL thread abstraction supports the creation and destruction of independent units of execution within the address space of a single process. Thread scheduling follows a simple priority scheme based on a discrete number of priority classes.

**[0043]** The OSAL semaphore abstraction supports intra-process, bounded and unbounded counting semaphores and the associated Wait/Signal operations that can be used to synchronise the execution state of different threads.

**[0044]** The OSAL file/directory abstractions provide a simple virtual file system with a hierarchical directory structure and read/write/execute file/directory permissions. The file system independent path names provide an operating system independent naming scheme that can be mapped onto different physical file system organisations. Operations are provided to create/delete files, read/write byte sequences from/to files, position on a particular byte number and retrieve a file's size. Operations over a directory include creation/deletion and iteration over the constituent files/directories.

**[0045]** The OSAL process abstraction may support the concept of virtual address spaces and code/data segments that need to be part of the address space. Operations may be provided to start/stop a process and to enquire about its status (running, terminated).

**[0046]** The OSAL mqueue abstraction may provide a means to perform inter/process communication based on byte oriented, first-in first-out message queues. Operations may be provided to put into or retrieve variable length byte strings from the queue and to check the total number of messages/bytes available on each queue.

**[0047]** Each of the OSAL abstractions may be supported by a different, strongly typed C++ handle class. The handle classes provide an object-oriented view on the operating system abstractions. Due to strong typing applications written on top of the OSAL, API benefit from stringent compile time type checking. Instances of handle classes represent operating system dependent entities and the operating system services are represented as methods on the handle class. The implementation of the handle classes is light-weight, efficient and portable across all platforms that support the ANSI C++ standard because it uses a low-level API whose implementation is operating system (Windows 95, Windows NT, Unix) specific. In most cases, the low-level API maps directly onto the services supported by the underlying operating system so that the overhead imposed by OSAL is minimal.

**[0048]** The OSAL class library and API may be used to build a cross-platform implementation of a DVB-compliant data broadcast server that supports the delivery of HTML, Java, MHEG applications and other data via DSMCC Object and Data carousels. The threading abstraction may be used to support (a) multiple threads generating the DSMCC object Carousel bit stream from content data, (b) a thread dedicated to generating DVB Service Information and (c) a

thread multiplexing DSMCC sections and table data into transport stream packets. The semaphore abstraction is used to implement the synchronisation between these threads. The file and directory abstractions are used to implement access to persistent content on the server's file system. The process abstraction is used to implement one process dedicated to network communication and another process handling the generation of the server's outgoing transport streams. The mqueue abstraction is used to implement the communication between the networking stack and the generation of the bit stream. Using the OSAL class library and API the data broadcast server application code runs unmodified on each of the operating systems that OSAL is currently supported on.

**[0049]** As explained above, each Stream Generator 26 may have a number of carousel builders 28. It may often be desired that the pages of one carousel be transmitted more often than the pages of another carousel. In order to do this, the multiplexer/packetiser 30 may introduce into the transport stream packets from one carousel more often than another. By way of example, where there are only two carousels and one has three times the weighting of the other, when filling transport stream packets, the multiplexer/packetiser assigns a 75% weighting to one carousel and will fill three packets with data from that carousel for every one packet filled with data from the other carousel.

**[0050]** Unfortunately, this system is computationally expensive when changes are made to the nature of the data transmitted on the transport stream. For instance, for the example given above, if a third carousel is added with a similar importance as the lower importance carousel already in use, then the system has to recalculate that the three carousels should be given respectively 60%, 20% and 20% of the transmission time. In other words, whenever the importance of the carousel is changed or carousels are added or deleted, the proportioning carried out by the multiplexer/packetiser 30 has to be recalculated.

**[0051]** In the information server, a new algorithm is proposed for the scheduling of the packets.

**[0052]** Referring to Figure 6, there is illustrated a data array for proportioning the importance of various data sources.

**[0053]** It is arranged as a series of notional time slots  $t$ . Each time slot  $t$  of the array allows the registration of one or more data sources, for instance carousels, raw data from storage 24 etc. By allocating each data source with a relative priority based on a notional frequency, the various data sources may be easily prioritised.

**[0054]** In the illustrated example, data source 1 is allocated as priority "4" and, hence, occurs every four notional time slots. Similarly, data source 2 is given a priority of "3" and data source 3 is given a priority of "2". The multiplexer/packetiser 30 then forms MPEG packets by following in sequence the data sources designated along the array, ignoring any empty notional time slots. For example, following the data sources illustrated in Figure 6, it fills a packet with data from data source 1, then a packet with data from data source 2, then a packet with data from data source 3, then a packet with data from data source 2, then a packet with data from data source 3, then a packet with data from data source 1 etc. In this way, data from data source 1 will be transmitted three times for every four times transmitted from data source 2 and for every six times transmitted from data source 3.

**[0055]** By using the array as described above, no recalculation is required when data sources are added or deleted or when the priority of a data source is changed. For instance, if data source 3 is deleted, the relative priority between data sources 1 and 2 is maintained. Similarly, if a fourth data source is transmitted with a very high priority, for instance allocation "1", then even though data sources 1, 2 and 3 are all transmitted then with half the frequency than before, their relative priorities and frequencies remain the same. Thus, the system of prioritising data operations is made computationally more efficient.

**[0056]** By using this system in the Stream Generators 26 of the described Information Server, it is then easy to allocate band width to various data sources, such as the various carousel builders 28. However, it is equally applicable to allocating processing time in a CPU. In particular, the various tasks requiring execution can be allocated with priorities in the same way as band width is allocated.

**[0057]** It is proposed that the ability to obtain resources, for instance processing or band width, need not be based only on a proportional share of the resources, but also on a latency of initial access. In other words, some processes may require action more immediately than others. For instance, the low latency allocation of CPU processing time is important in ensuring smooth operation of multi-media applications. Where two processes, for instance word processing and video display, are allocated a preset proportion of processing time, it may still be that it is more important for the video display to be given more immediate access to the processing.

**[0058]** In order to account for this need, an additional parameter may be assigned to each resource consumer or data operation, whether this is data processing or transmission of data. This parameter is intended to define the latency of resource access.

**[0059]** Thus, each resource or data operation is given a latency value, such that a number of resource consumers may be compared in order to determine which should be conducted first or, similarly, a number of resource consumers may be compared to see which should be transmitted first. In the case of the Information Server discussed above, where there is a very important piece of information which should be transmitted and received as soon as possible, the appropriate data source can be given a very high latency value, such that this data will be inserted by the multiplexer/packetiser 30 into the data stream as soon as possible.

**[0060]** In the carousel builders 28 described above, the carousel builders each construct a plurality of DVB sections

for supply to the multiplexer/ packetiser 30. These sections include data representing the carousel of that particular carousel builder. It is desired that the Information Server or the carousel builder itself be able to change or replace one or more of the pages within its carousel. However, as explained above, once a carousel builder 28 has constructed an appropriate carousel from the data in storage 24, that data is not recalculated, but is merely stored for cyclical transmission. In order to change a page or file in the carousel, it is possible to recalculate all of the data for that carousel. However, this can result in an undesirable delay before the new carousel is transmitted.

[0061] Figure 7 illustrates the process of data conversion conducted by the carousel builder. The data files have headers added to them to form BIOPs as part of a DSMCC object carousel. These are then grouped together to form Modules for the DSMCC data carousel, which are then in turn divided into DVB sections for passing on to the multiplexer.

[0062] As will be appreciated, if File 1 is changed, then the BIOP File 1 will also be changed, together with the resulting module and sections. However, the BIOPs for Files 2 and 3 will remain unchanged. Therefore, after a change to File 1, it is not necessary for the carousel builder to recalculate BIOPs for Files 2 and 3.

[0063] In order to enable the carousel builder 28 to avoid the unnecessary calculation, a dependency mechanism is proposed. By using this mechanism, it is possible to rapidly update the bit stream representation of a DSMCC object carousel by only using updates to its constituents, i.e. files, directories etc..

[0064] The dependency structure is arranged in the carousel builder by a series of nodes and pointers. In particular, the dependency network consists of two types of entities, namely (a) dependency nodes that store pointers, called content references, to particular sub-parts of the bit stream representation of the DSMCC object carousel and (b) dependency links that indicate how one set of content references is used to calculate another set of content references. The dependency nodes themselves come in two types, namely (1) primary nodes that contain the content references to the constituents (e.g. the DSMCC file and directory objects) of a DSMCC object carousel and (2) computed nodes that contain transformations to the content references of other nodes. A transformation is applied to a set of content references and results in a set of new content references pointing to new parts of the DSMCC object carousel bit stream.

[0065] As a carousel builder constructs the appropriate data, dependency links are stored so as to indicate how one piece of the DSMCC object carousel bit stream (e.g. a module) leads to another piece of the bit stream (e.g. the compressed equivalent of the same module). The dependency links are established when transformations are executed by the computed nodes of the dependency network. In particular, when a transformation of a dependency node uses the content references associated with another node (whether a primary node or a computed node), a dependency is established between the used node and the one that calculates the transformation. A dependency link is stored with respect to the used node indicating the node for which it was used.

[0066] The dependency links allow the dependency mechanism to determine quickly the set of transformations that need to be reapplied in order to recalculate a new bit stream when constituents of the object carousel change, i.e. when data of the primary nodes change.

[0067] When an update is carried out, some of the DSMCC file and directory objects will be changed. As a result, the primary nodes relating to those changes will also be changed. For those primary nodes where a change has occurred, the dependency links associated with those primary nodes are followed to mark any resulting computed node. Where a computed node is reached, the dependency link of that computed node is used to move forward again to any further computed node. All of the computed nodes which are reached in this way are cleared, together with any forward dependency links. As a result, the dependency network has certain linked computed nodes empty. The carousel builder therefore repeats the original calculation process for those computed nodes which are now empty. As a result, when the computed nodes reapply their transformations to recompute a new set of content references, dependency links are once again established from the nodes to which they have referred.

[0068] Where primary nodes have not been changed or computed nodes not been deleted, then a subsequent computed node can merely refer back to the unchanged primary or computed nodes. Thus, if a computed node refers only to unchanged primary computed nodes, it can be used directly without reapplying its transformation. As long as the content references on which the result depends do not change, the result can be reused in an updated bit stream directly without having to recompute it. Since the parts of a carousel which are not affected by an update are already stored in the dependency network, the dependency network allows for rapidly updating the content of a DSMCC object carousel.

[0069] In this way, the dependency mechanism may be used to build up bit stream representations of DSMCC sections that make up object carousels. The bit stream representation consists of a list of content references pointing to the different sections. The content references are passed to the multiplexer that packetises the DSMCC sections into an MPEG transport stream. The mechanism may be used to recompute the modified DSMCC sections when constituent parts of the DSMCC object carousel (e.g. files and directories) change. It is also possible to apply the technique to DSMCC data carousels.

[0070] This is illustrated in Figure 8. In particular, by changing the primary node n4 and referring to the dependency link, it is evident that the content of computed node n7 must also be changed. By referring to the dependency link for

computed node n7, it is then also clear that the content for computed node n8 must also be changed. With the content of computed nodes n7 and n8 deleted and with primary node n4 containing the new content references, computed node n7 is recalculated using the content references of primary nodes n4 and n5. The content references for computed node n8 may then be recalculated from content references of computed nodes n6 and n7. It will be seen that, in this process, it is not necessary to recompute the content references for computed node n6.

[0071] For each Stream Generator 26 of the Information Server, it is desirable to provide some further flexibility in updating the DSMCC file objects to be transmitted by the transport stream.

[0072] As discussed above this Stream Generator 26 includes one or more carousel builders 28 and a multiplexer/packetiser 30 for constructing the resulting MPEG transport stream. Furthermore, a dependency network is built into the process of computing the required DSMCC sections of a DSMCC object carousel, called payloads, that constitute the bit stream at DSMCC level.

[0073] It is desired to provide a "hot insertion" of DSMCC file objects. This is an out-of-band insertion of a DSMCC file object as soon as possible in the broadcast bit stream.

[0074] As explained above, once the required DSMCC sections of a DSMCC object carousel have been calculated for a carousel, they are merely stored and retransmitted cyclicly. It would not, therefore, be desirable to merely hot insert a DSMCC file object into the objects of the carousel, since the DSMCC level bit stream would then have to be recalculated. On the other hand, merely inserting DSMCC sections for a hot insert into the already established bit stream is likely to corrupt transmission of the DSMCC object carousel.

[0075] The multiplexer repeatedly loops through the list of computed payloads and dynamically chops up the payloads into MPEG2 transport stream packets for transmission. In order to allow for "hot insertion" of DSMCC file objects, "insertion points" are defined inside the payload list of the complete set of DSMCC sections for the object carousel. These "insertion points" are markers where the multiplexer can safely insert additional payloads for an out-of-band DSMCC file object.

[0076] Thus, the payload for the out-of-band update is computed and passed to the multiplexer. The multiplexer then waits until the next available marked "insertion point" and then inserts the out-of-band payload. The effect of this is to delay transmission of the rest of the carousel data but not to interfere with its content.

[0077] As mentioned above, it is also desirable to provide a cyclic update of a single DSMCC file object. In other words, for every round of the carousel broadcast, a different version of a particular DSMCC file may be transmitted.

[0078] With regard to the dependency network described above, the cyclic update is achieved by having a separate node in the dependency network that is used to compute the payloads for each version of the DSMCC object. The complete list of payloads is presented to the multiplexer, but this list is organised internally in such a way that it delivers the next payload value whenever the multiplexer requests it in the next cycle of the object carousel. This mechanism is extremely efficient, since, rather than calculating an updated version of the DSMCC object for each round of the carousel, a pointer may merely indicate the appropriate payload.

[0079] It is desirable that the selection of DSMCC file objects with regard to broadcast order, broadcast frequency and hot update is implemented by the user. In order to do this, it is proposed that data should be entered in the manner of the table given below.

Table 1

| Reference Name | Attribute | Full Name        |
|----------------|-----------|------------------|
| page 1         | Normal    | /sports/intro    |
| page 2         | Cyclic    | /news/item 1     |
|                |           | /news/item 2     |
|                |           | /news/item 3     |
|                |           | /news/item 4     |
| page 3         | Immediate | /gambling/horses |
| page 2         | Cyclic    | /news/item 5     |
|                |           | /news/item 6     |

[0080] As may be seen, the table consists of three columns. The first column lists the reference name of a DSMCC file object as it will be handled by the receiver of the broadcast. The complete list of reference names constitutes the virtual name space that the receiver handles. The second column specifies the attribute for the item referenced by the reference name. It is proposed to provide three attributes, namely normal, cyclic and immediate. These will be described below. The third column specifies a list of full names, in other words a physical file name.

[0081] The attribute "normal" means that the referenced page will always output the listed file. On the other hand, for the attribute "cyclic", the table may contain more than one full name, such that in each cycle of the carousel, the



next full name listed for that page will be included. The attribute "immediate" normally has the same effect as "normal" in that a single file will be transmitted for that page. However, where a referenced page has the attribute "immediate" as soon as an update is received for that page, the page will be inserted as a "hot insert", but will then not again appear until its normal position in the carousel.

[0082] For the example given above, the carousel configuration table defines a logical carousel consisting of 3 pages named: page 1, page 2, page 3. The broadcast order is defined by the first column, i.e. -page 1-page 2-page 3-page 2 and then back to page 1. However, for the cyclic pages in each cycle of the carousel the next item from the list of full names (third column will be taken). Consequently, from a content point of view the carousel order will be: /sports/intro-/news/item 1-/gambling/horses/news/item 5-/sports/intro-/news/item 2-/gambling/horses-/news/item 6-/sports/intro-/news/item 3-/gambling/horses-/news/item 5-/sports/intro/news/item 4-/gambling/horses-/news/item 6...

[0083] Following the entry of the data for the carousel configuration table, an appropriate carousel builder computes the required DSMCC section according to the dependency network discussed above. In particular, the pages defined by the reference names are set up as primary nodes. Where a page is defined as being cyclic, then the carousel builder sets up equivalent duplicate pages corresponding to each object of the cycle. The carousel builder then computes appropriate modules for the pages. Preferably, in computing the modules, equivalent cyclic pages are assigned to separate modules, such that at each cycle recalculation is kept to a minimum. Similarly, pages which have the attribute "immediate" are assigned to separate modules so as to facilitate their insertion when necessary.

## Claims

1. An information server (18) for outputting a carousel of files in a transport stream, the information server comprising:

at least one carousel builder (28) for transforming data from the files through a protocol stack to form transport stream packets;  
a series of registers; and  
a dependency mechanism which, for each file used in a transformation, stores a dependency link indicating the transformed data resulting from the transformation and, for any transformed data used in a transformation to form further transformed data, stores a dependency link indicating the further transformed data such that, when a file in the carousel is changed, the dependency links stored in the registers indicate any transformed data requiring recalculation as a result of the change.

2. An information server according to claim 1 wherein the registers form part of a dependency network comprising dependency nodes, the dependency nodes including primary nodes that are based on files of the carousel and computed nodes that are based on transformed data at successive layers of the protocol stack.

3. An information server according to claim 2 wherein the dependency nodes store pointers to particular parts of a bit stream representation of the carousel of files.

4. An information server according to claim 3 wherein the dependency links are stored as part of the dependency network and indicate how one part of the bit stream is used to calculate another part of the bit stream, the pointers acting as content references to the bit stream.

5. An information server according to claim 1, 2 or 3 wherein the carousel of files comprises a DSMCC object carousel.

6. An information server according to claim 4 wherein the carousel of files comprises a DSMCC object carousel and wherein the content reference pointers of the dependency nodes point to particular sub-parts of the bit stream representation of the DSMCC object carousel and the primary nodes contain the content references of the DSMCC file and directory objects.

7. An information server according to any preceding claim wherein at least one of said files comprises multiple versions, a different version being transmitted on consecutive rounds of the carousel broadcast;

the or another at least one carousel builder (28) transforming the data of each of said versions;  
the dependency structure storing dependency links for each version; and  
consecutive versions of the transport stream packets being selected upon each round of the carousel broadcast.

8. A method of constructing a transport stream from a carousel of files, the method comprising:

transforming data from the files through a protocol stack to transport stream packets;  
for each file used in a transformation, storing a dependency link indicating the transformed data resulting from  
the transformation and, for any transformed data used in a transformation to form further transformed data,  
storing a dependency link indicating the further transformed data, such that, when a file in the carousel is  
changed, the dependency links indicate any transformed data requiring recalculation as a result of the change.

5

10

15

20

25

30

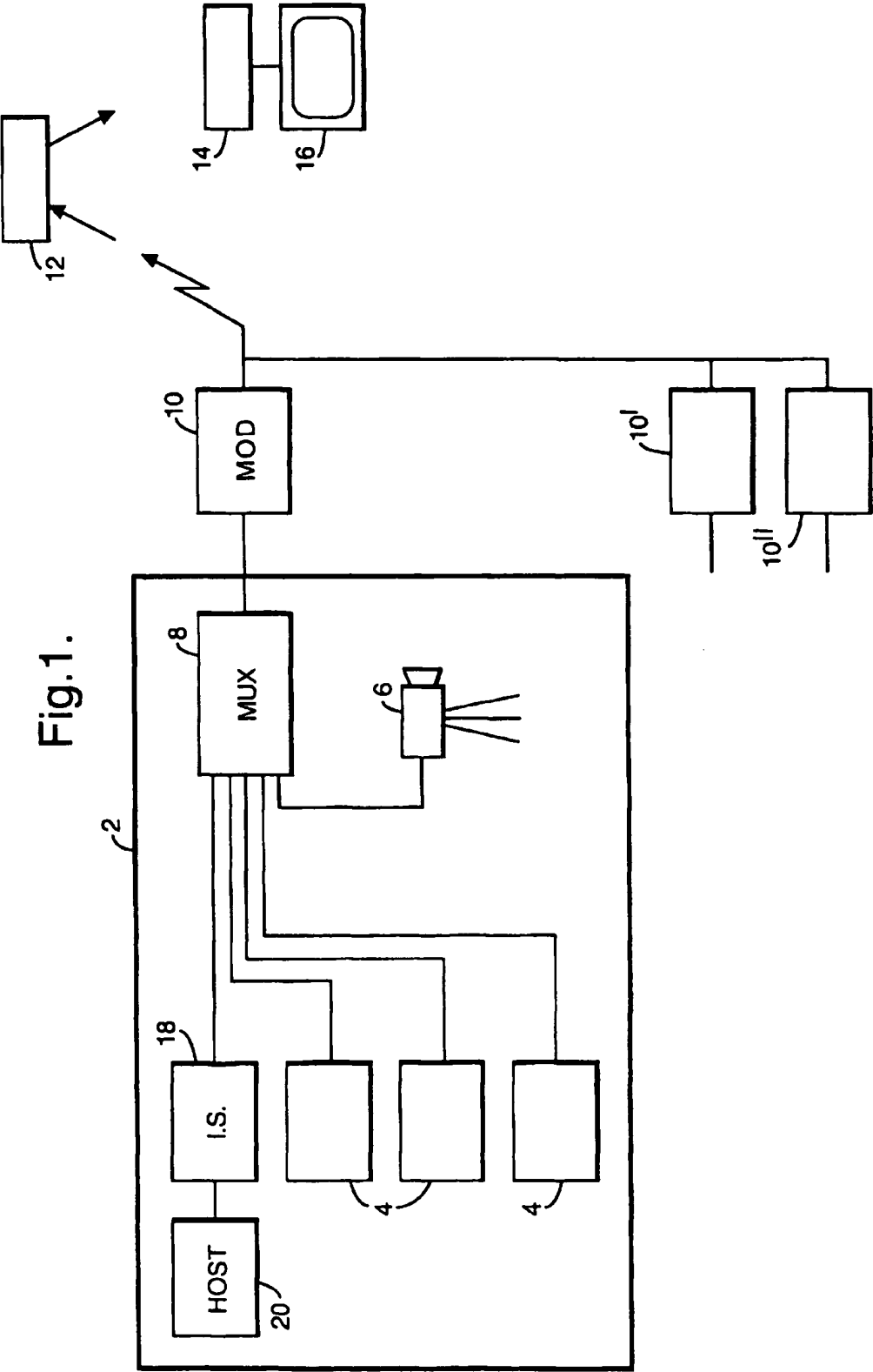
35

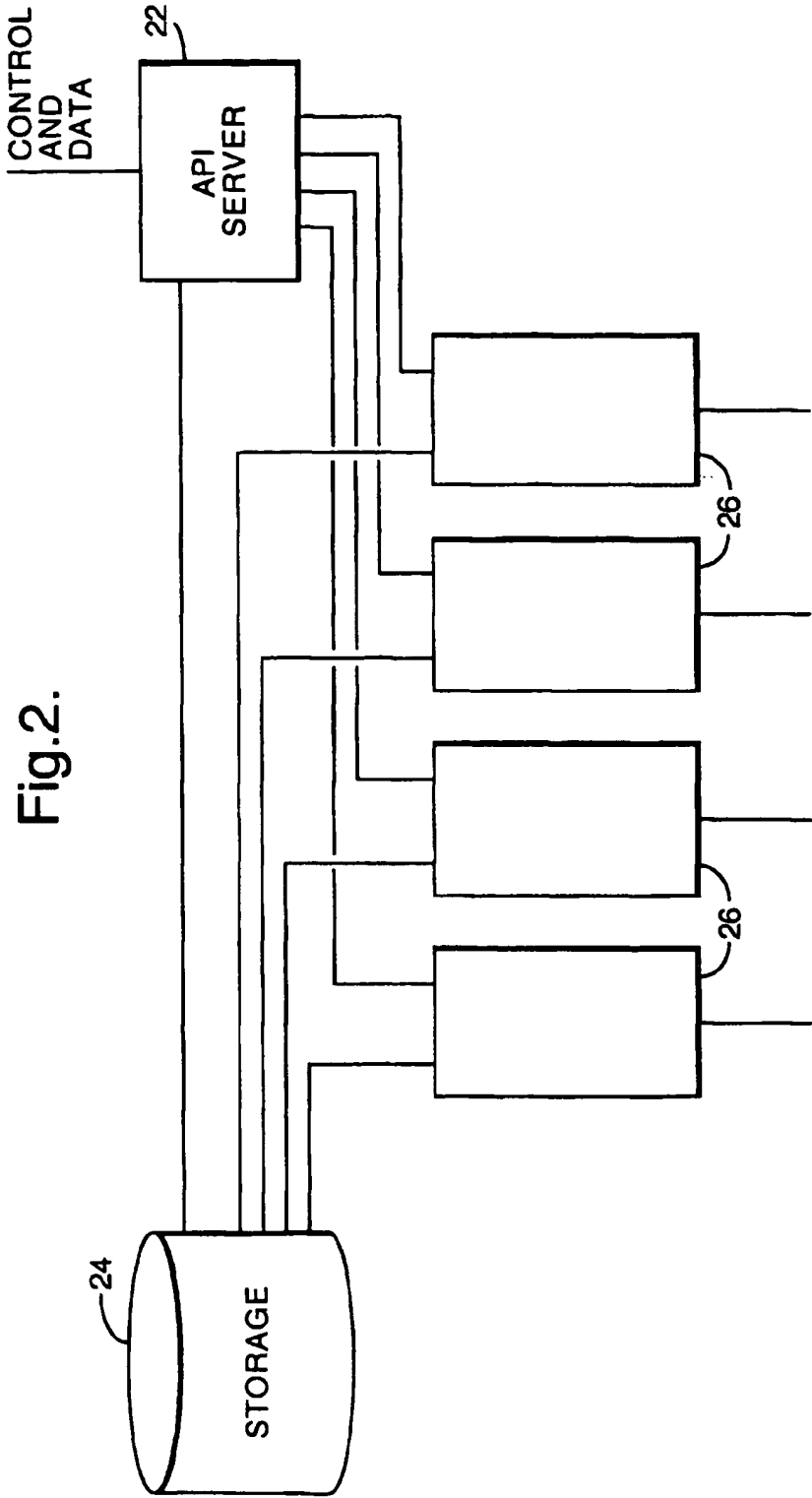
40

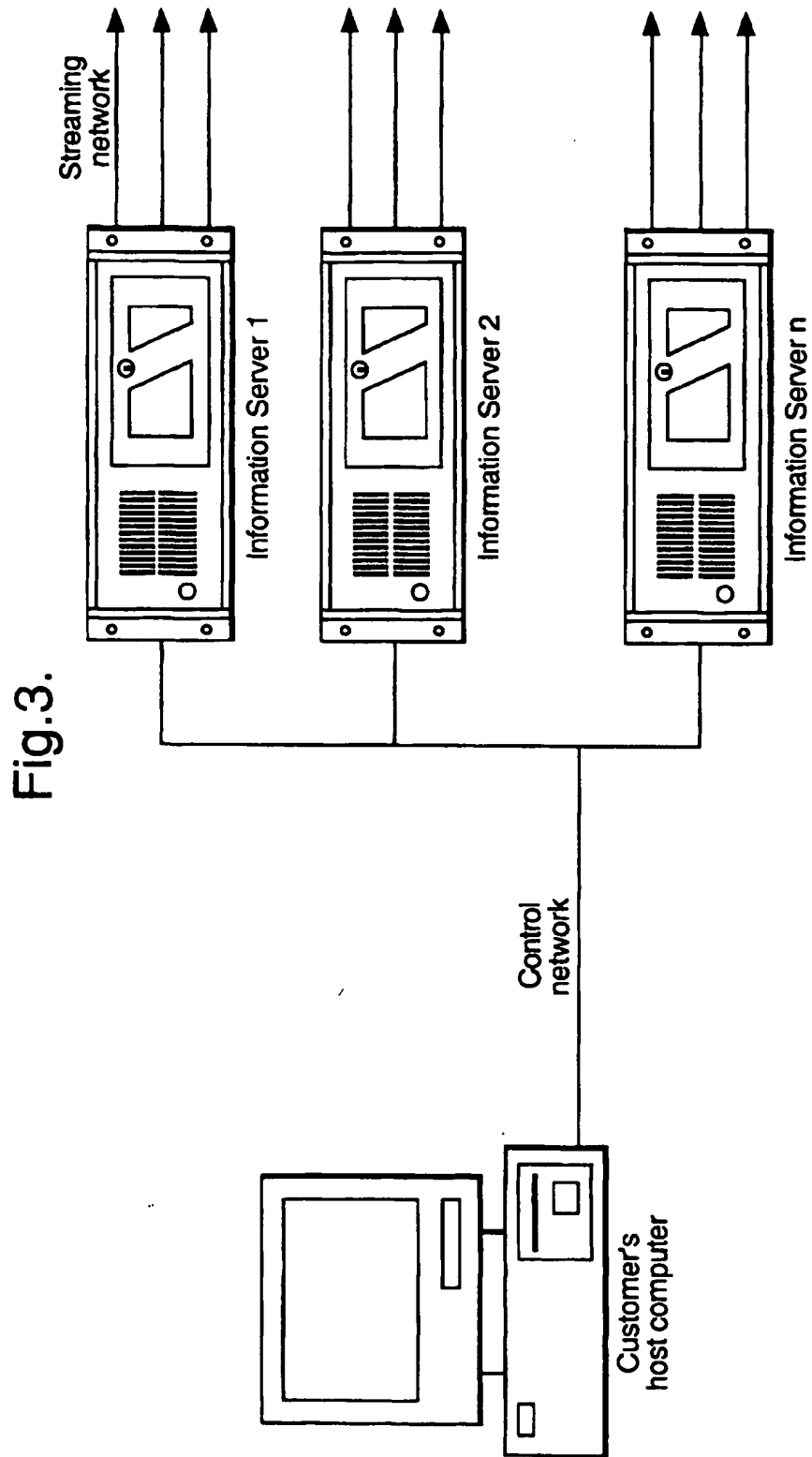
45

50

55







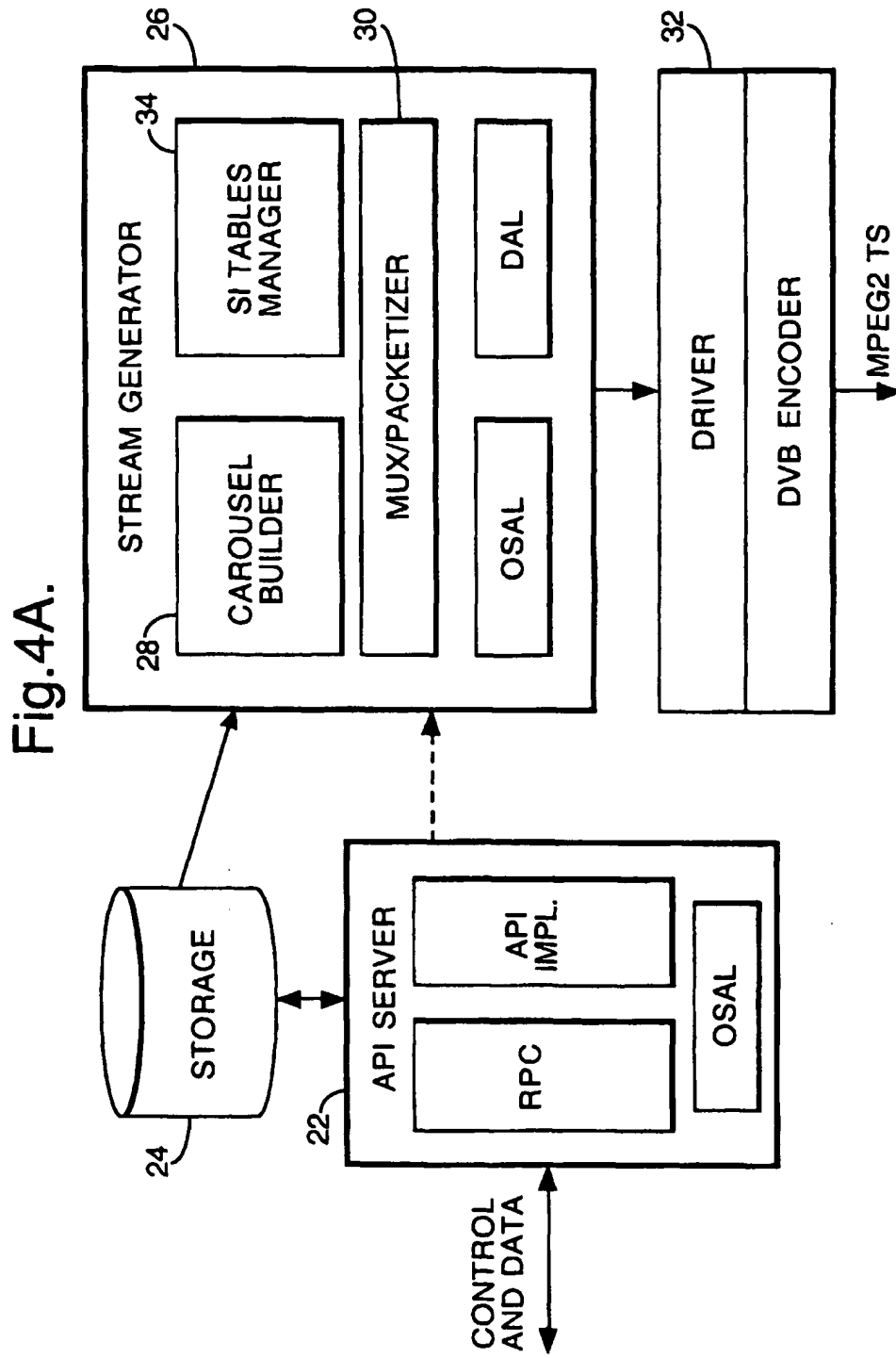
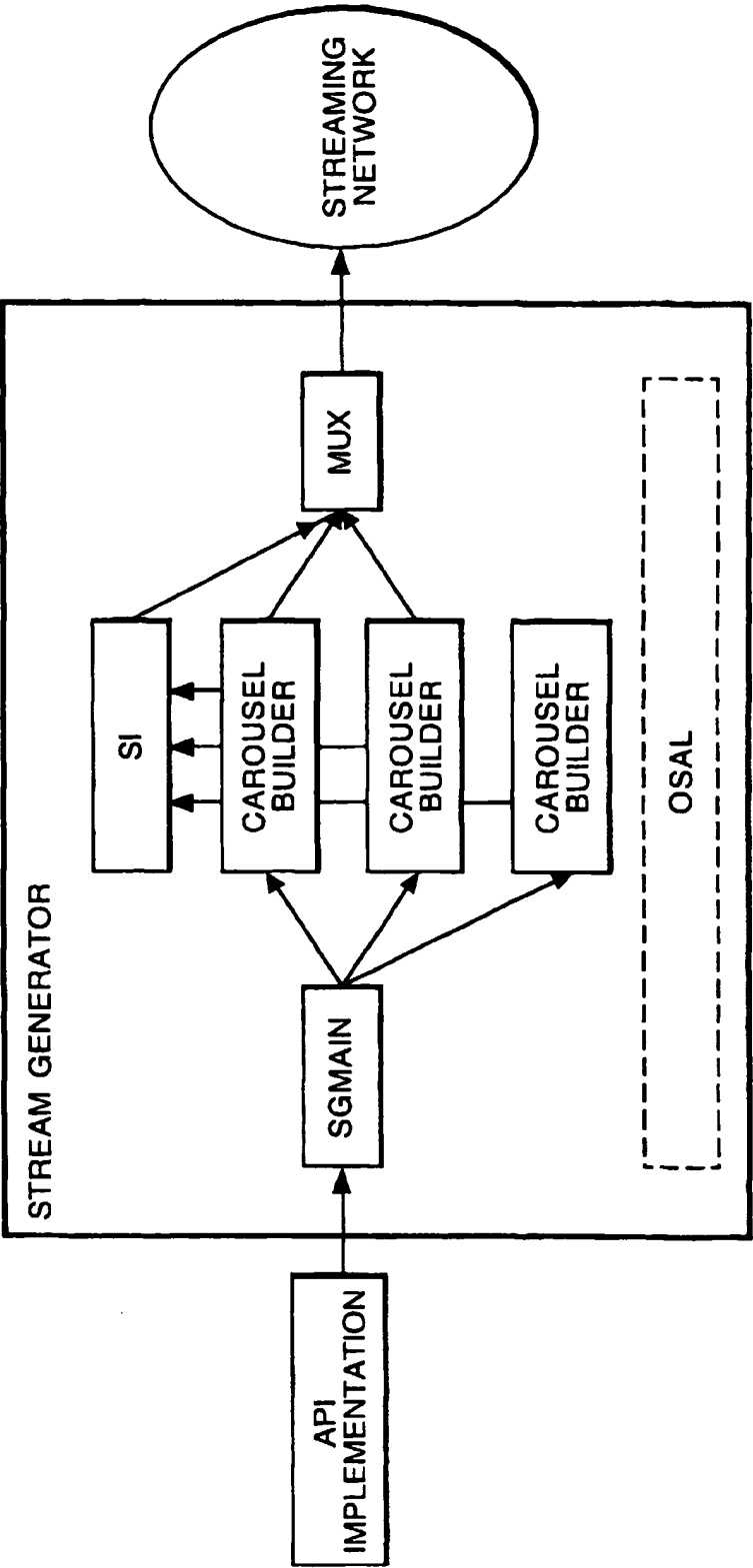


Fig.4B.



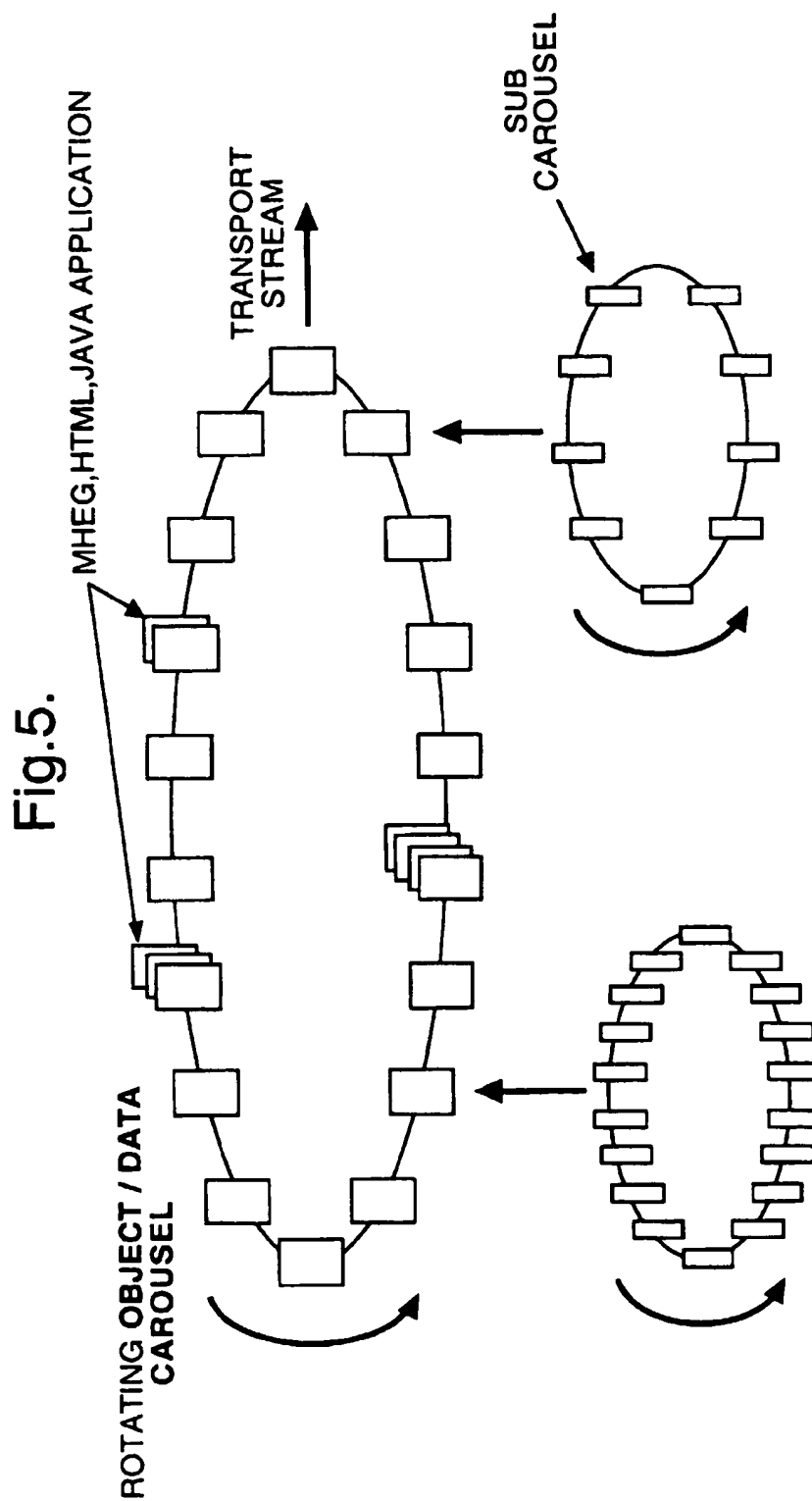




Fig.6.

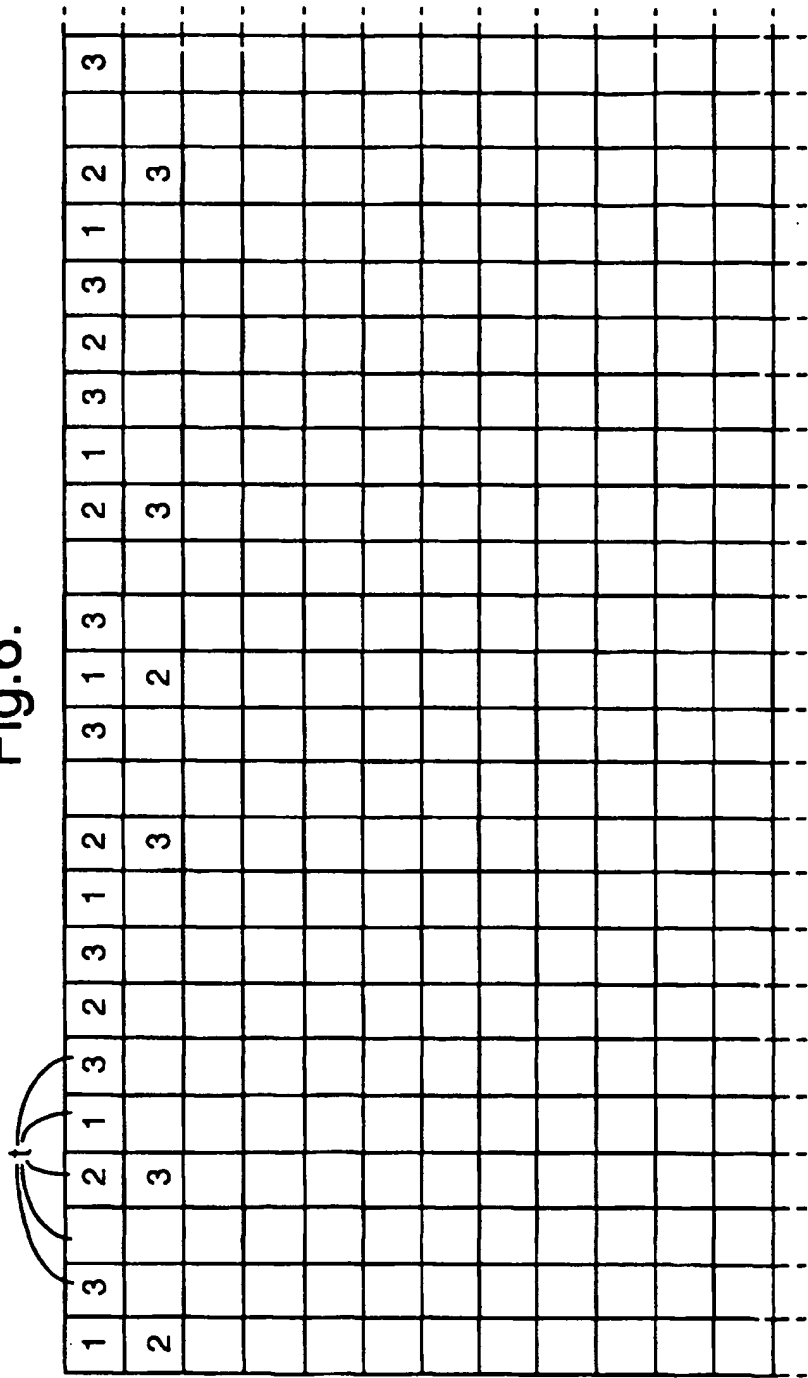


Fig.7.

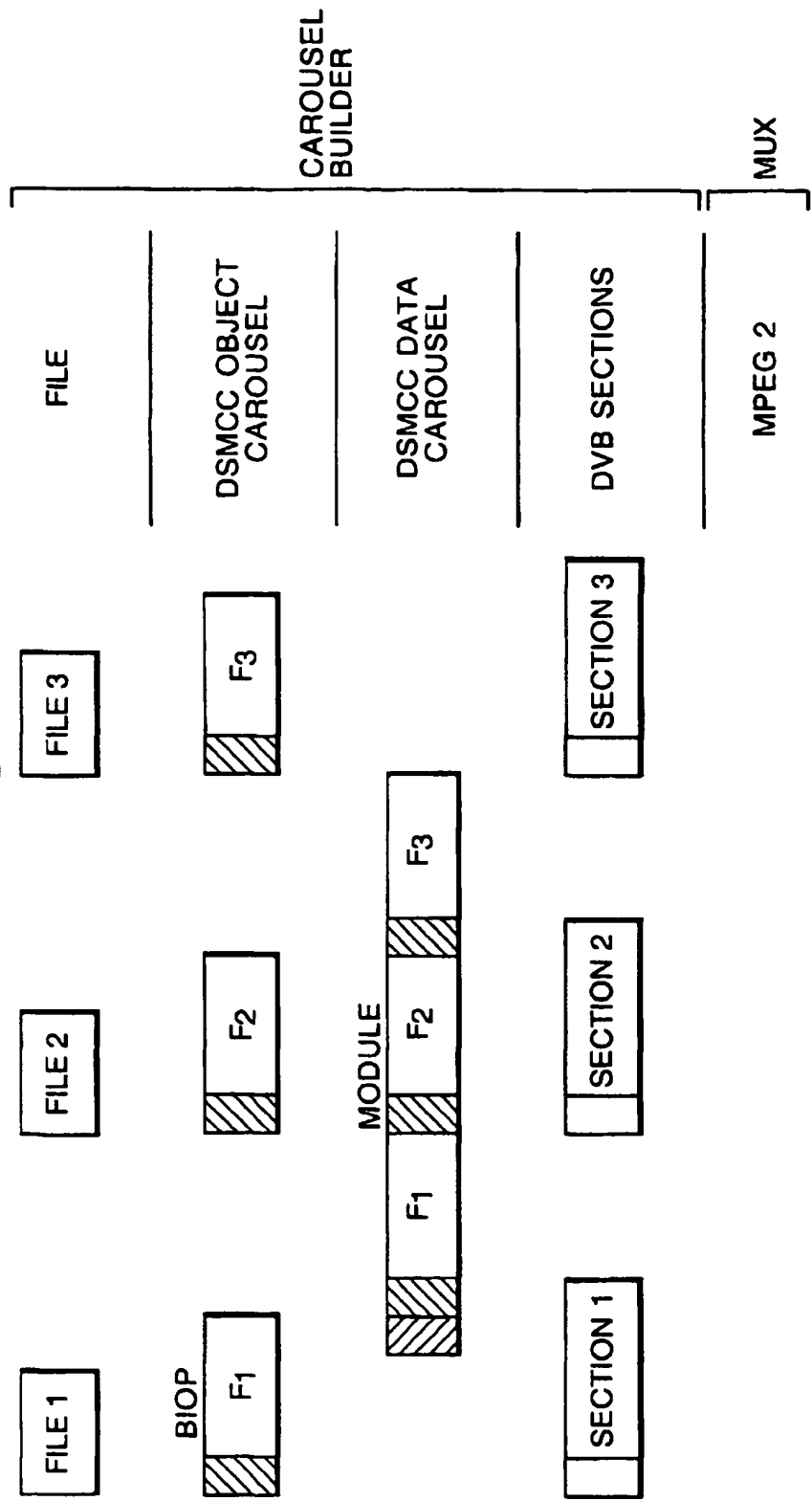
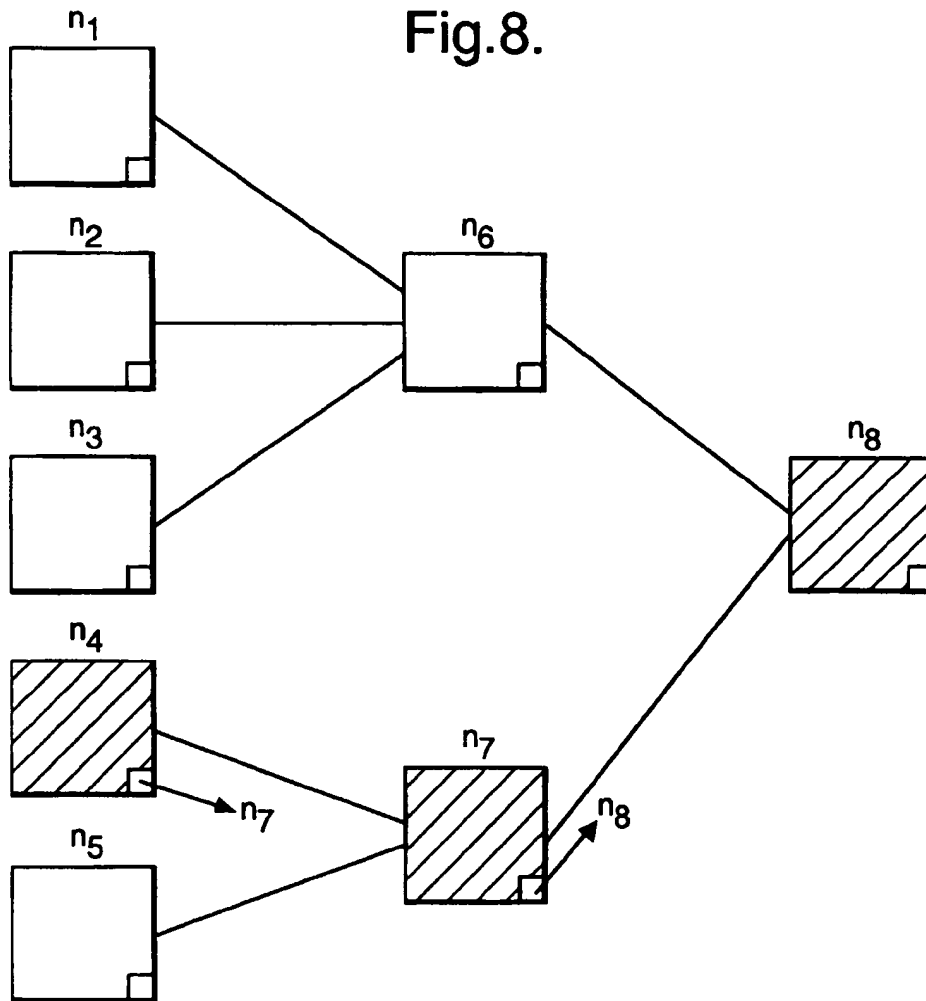


Fig.8.





European Patent  
Office

# EUROPEAN SEARCH REPORT

Application Number  
EP 99 30 0438

| DOCUMENTS CONSIDERED TO BE RELEVANT   |   |  |  |
|---|---|--|--|
| Category  | Citation of document with indication, where appropriate, of relevant passages   | Relevant to claim                                | CLASSIFICATION OF THE APPLICATION (Int.Cl.6) |
| A   | US 5 805 825 A (DANNEELS GUNNER D ET AL)<br>8 September 1998<br>* column 3, line 18 - column 6, line 23;<br>figures 2-4 *                             | 1,8  | H04N7/24                                     |
| A   | WO 90 06647 A (COMPUQUEST INC)<br>14 June 1990<br>* abstract; figures 2,7 *<br>* page 10, line 22 - page 12, line 30 *                                | 1,8  |  |
| A   | EP 0 854 650 A (NOKIA TECHNOLOGY GMBH)<br>22 July 1998<br>* page 3, line 50 - page 4, line 10 *<br>* page 6, line 56 - page 7, line 55;<br>figure 6 * | 1,8  |  |
|   |   |  | TECHNICAL FIELDS SEARCHED (Int.Cl.6)         |
|   |   |  | H04N   |
| The present search report has been drawn up for all claims  |   |  |  |
| Place of search<br>BERLIN   |   | Date of completion of the search<br>21 June 1999 | Examiner<br>Deane, E                         |
| <p>CATEGORY OF CITED DOCUMENTS</p> <p>X : particularly relevant if taken alone<br/>Y : particularly relevant if combined with another document of the same category<br/>A : technological background<br/>O : non-written disclosure<br/>P : intermediate document</p> <p>T : theory or principle underlying the invention<br/>E : earlier patent document, but published on, or after the filing date<br/>D : document cited in the application<br/>L : document cited for other reasons<br/>&amp; : member of the same patent family, corresponding document</p> |   |  |  |

EPO FORM 1503 03.92 (P04C01)

**ANNEX TO THE EUROPEAN SEARCH REPORT  
ON EUROPEAN PATENT APPLICATION NO.**

EP 99 30 0438

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report.  
The members are as contained in the European Patent Office EDP file on  
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

21-06-1999

| Patent document<br>cited in search report | Publication<br>date | Patent family<br>member(s)   | Publication<br>date      |
|---|---------------------|------------------------------|--------------------------|
| US 5805825 A                              | 08-09-1998          | NONE                         |                          |
| WO 9006647 A                              | 14-06-1990          | US 5010553 A<br>CA 2004536 A | 23-04-1991<br>05-06-1990 |
| EP 0854650 A                              | 22-07-1998          | FI 970186 A                  | 17-07-1998               |

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82